LERO – THE IRISH SOFTWARE ENGINEERING RESEARCH CENTRE

MODEL DRIVEN THEME/UML

Eclipse Theme/UML-Plugin Manual

Author:
Kai BECKMANN
Éamonn LINEHAN

5th February 2009

The Theme/UML-Plugin User Manual

Copyright © Distributed Systems Group, Department of Computer Science, Trinity College Dublin 2009. All rights reserved.

Comments may be addressed to:

Dr. Siobhán Clarke
Distributed Systems Group,
Department of Computer Science,
Trinity College,
Dublin 2.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

All code listings in this document are part of the Theme/UML-Plugin. The Theme/UML-Plugin is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; version 3 of the License.

The Theme/UML-Plugin is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. A copy of the GNU Lesser General Public License is distributed with the Theme/UML-Plugin. It can be found in a file named COPYING.txt. If not, see <http://www.gnu.org/licenses/>.

Contents

1 Introduction to the Theme/UML-Plugin 1
1.1 Files . . . 2
1.2 System Requirements . . . 2

2 Installation 3

3 Using the Theme/UML-Plugin 6
3.1 Composing the Theme/UML model . . . 6
3.2 PIM2PSM . . . 6
3.3 Code-Generation . . . 7
3.4 Limitations . . . 8
3.4.1 Code-Generation . . . 10
3.4.2 ANSI-C Code-Generation . . . 11

4 OOP-Model for C 11

1 Introduction to the Theme/UML-Plugin

The Theme/UML Plugin helps you use the model driven Theme/UML approach for software development (Figure 1).

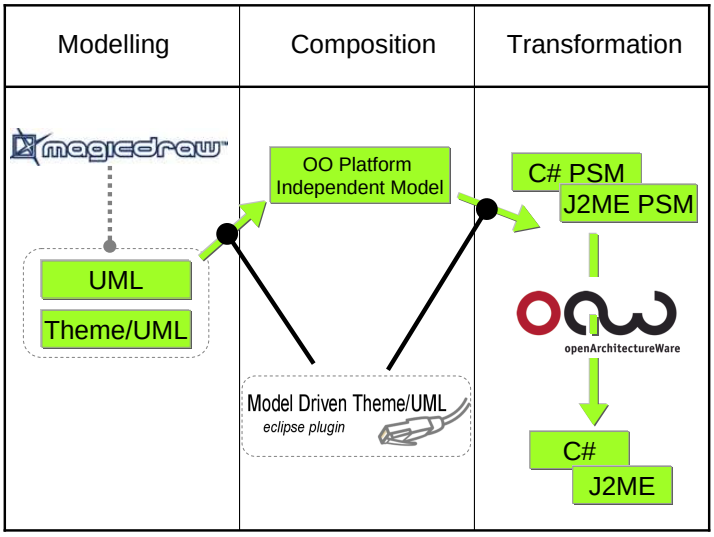

Figure 1: Model driven development process.

Figure 1 illustrates the development process. The Modelling phase involves producing a AO model of the program you wish to build. This model must be compatible with Theme/UML. The output from this phase is a collection of UML and Theme/UML documents stored in the XMI 2.0 format. We have used MagicDraw 15.0 to produce these files but any XMI compatible modelling tool should work. The composition phase uses the eclipse Theme/UML plugin to perform composition of the AO and OO base model, resulting in a Platform Independent Model (PIM). In the transformation the PIM is transformed into a Platform Specific Model (PSM) via the plug-in. This model is then transformed into source code using OpenArchitectureWare.

The Theme/UML-Plugin supports three phases of model driven development:

1. The composition of the aspect oriented model to “normal” UML.
2. The transformation of the platform independent model to a platform specific one.
3. The transformation of the platform specific model to text (that is, the generation of source code).

The code generation is done by a couple of OpenArchitectureWare eclipse projects which are provided.

1.1 Files

The following files are part of the Theme/UML-plugin and can be downloaded form the Theme/UML-Plugin homepage.

handbook.pdf The Theme/UML-Plugin manual.

ThemeUMLPlugin_1.0.3.jar The eclipse plugin jar of the Theme/UML-Plugin.

oaw_project_ansi-c.zip The OpenArchitectureWare project for the ANSI-C target platform.

oaw_project_aspectj.zip The OpenArchitectureWare project for the aspectj target platform.

oaw_project_j2me.zip The OpenArchitectureWare project for the j2me target platform.

oaw_project_netcf.zip The OpenArchitectureWare project for the .NET target platform

uml_example_pacemaker.zip An example for an UML project, modeling a pacemaker.

uml_example_tollsystem.zip An example for an UML project, modeling a device for an toll system.

theme-uml-plugin-source-code.zip The source code of the Theme/UML-Plugin.

ThemeUMLPlugin-JavaDoc.zip The javadoc API documentation for the Theme/UML-Plugin.

1.2 System Requirements

For producing the UML models a EMF XMI 2.0 compatible modelling tool is required (we recommend MagicDraw). The Model Driven Theme/UML tool requires an eclipse (Ganymede v3.4.0).

2 Installation

The Model Driven Theme/UML tool consists of an eclipse plug-in and a couple of OpenArchitectureWare projects.

The Theme/UML-Plugin has the following eclipse dependencies:

- Java 1.6 JDK
- EMF
- UML2
- OCL
- log4j

The first step is to download and install the current JDK (Java Development Kit) from the Sun homepage[1] (Windows user) or install it by using your package management system (Linux/Unix user).

We recommend to download a fresh “Ganymede” eclipse zip archive (Version 3.4.0 or higher) with “Modeling Tools” included[2]. After the unpacking the log4j-plugin has to be installed. This can be done by the eclipse software update function using the following steps[3]:

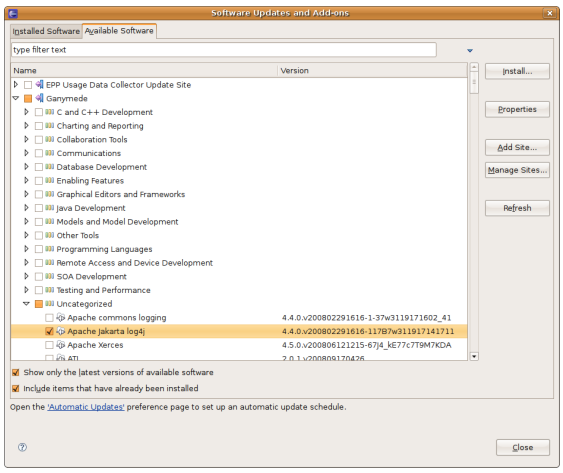

Figure 2: Select log4j from the eclipse update service.

[1] http://java.sun.com/javase/downloads/index.jsp
[2] http://www.eclipse.org
[3] If your computer is behind a proxy, this have to be configured first:

- Click on “Window/Preference”
- Set your proxy under “General/Network Connections”

1. Click on “Help/Software Updates”
2. Click on “Available Software”
3. The log4j-plugin is under “Ganymede/Uncategorized” (see Figure 2)
4. Select “Install”.

See figure 2 for the right option to select at the eclipse update program.

Now you can copy the “ThemeUMLPlugin_1.0.3.jar”-file into the plugin folder of your eclipse installation. Under windows this could be `C:\Program Files\eclipse\plugins`; under Linux the path might be `/usr/lib/eclipse/plugins/` or the place where the eclipse zip archive was unpacked. After this step you have to restart eclipse. The plugin is now available in the context-menu when you right-click on an UML-file (Figure 3).

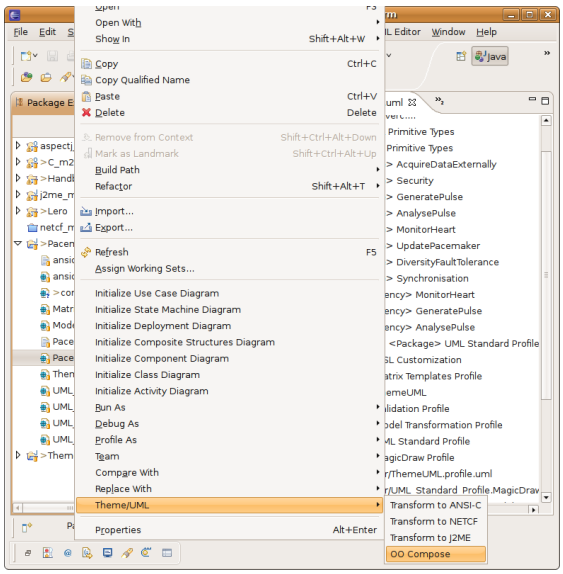

Figure 3: The Theme/UML-plugin menu entry shows up, after the installation of the plugin

There are two example projects provided, the “Tollsystem” and the “Pacemaker”. These projects can be downloaded and imported into eclipse. These projects contain UML-files with the Theme/UML approach. The Theme/UML-Plugin can compose those into normal UML files.

I addition to the Theme/UML-Plugin there are four OpenArchitectureWare projects provided. These projects can be used to generate source code from the composed UML-files. These projects depend on the OpenArchitectureWare eclipse plugins, that have to be installed separately. To do this the following steps are necessary:

To add the OpenArchitectureWare update site to the eclipse software update system:

1. Click on “Help/Software Updates”
2. Click on “Available Software”
3. Click on “Add Site”
4. Add the URL: “http://www.OpenArchitectureWare.org/updatesite/milestone/site.xml”
5. The new update site is added to the overview.
6. Select the combobox of the OpenArchitectureWare and click on install.
7. Restart eclipse.

You can now import the provided OpenArchitectureWare projects into your workspace. The classpath of these project have to be set manually. Right-click on each project and select the “Properties” option. Go to the “Java Build Path” window and remove all entries listed in the “Library” tab, except the J< RE.

If you have removed the JRE entry as well add it again by clicking on “Add Library” and chose there “JRE System Library” and click on the “Next” button.

The next step is to add all necessary libraries and plugins. The files can be found in the plugin folder of the current eclipse installation. Click on the “External Jars” button, brows to the plugin folder of your eclipse installation and select the following jar files (only the prefix is listed):

- org.eclipse.emf.common
- org.eclipse.uml2.uml
- org.eclipse.emf.ecore
- org.openarchitectureware.core.workflow
- org.apache.commons.cli
- org.apache.commons.logging
- org.openarchitectureware.core.xpand2
- org.openarchitectureware.core.expressions
- org.openarchitectureware.core.emftools
- org.eclipse.emf.ecore.xmi
- org.openarchitectureware.uml2.adapter
- org.eclipse.emf.mapping.ecore2xml
- org.eclipse.uml2.uml.resources
- org.antlr
- org.eclipse.uml2.common

3 Using the Theme/UML-Plugin

The following section describes the composing and transforming phases of model driven Theme/UML. The three design phases are described separately, but each phase builds on the output of the previous one. For an introduction to Theme/UML see the Theme/UML book[4].

3.1 Composing the Theme/UML model

After the installation of the Theme/UML-Plugin in the eclipse plugin folder there is a Theme/UML entry on the context menu of an *.uml file.

If a new project is started, a new eclipse project for the UML files should be created as well. After the modeling process with the UML-tool the diagrams have to be exported as XMI encoded files (EMF UML2 (v2.x) XMI) into the eclipse project folder.

The export creates a couple of *.uml files, the important one is the with the project name in it. This file can be manually edited in eclipse with a XML editor. Minor changes and fixes can be done this way easily.

The composition of the aspect themes to a “normal” UML model is initiated with a right click on the `<project name>.uml` file and a click on the “OO Compose” option in the Theme/UML menu. A new file will be show up in the eclipse project folder, called `composed_<projectname>.uml`. Sometimes eclipse has a problem opening this file, in this case select the refresh option from the context menu or press F5.

The new generated UML file can be edited with the XML-editor or the model can be viewed and modified with the eclipse UML editor. For this the option “Initialise Class Diagram” has to be chosen from the context menu of this file.

3.2 Platform independent model to platform specific model transformation

There are four different supported target platforms to generate a platform specific UML model: aspectj, j2me, .NET and ANSI-C. The transformation is done by the plugin. To start the transformation mark the composed UML-file from the last step and chose the “Transform to <target>“ option (replace target with you intended target platform), see Figure 4.

A new file will be created called “<target name>_composed_<project name>.uml”. This file contains the PSM for the select target platform. Like all UML-files this can also be viewed and modified with the eclipse XML editor or the graphical UML tool.

[4] Siobhán Clarke and Elisa Baniassad “Aspect-Oriented Analysis and Design – The Theme approach” Addison-Wesley 2005

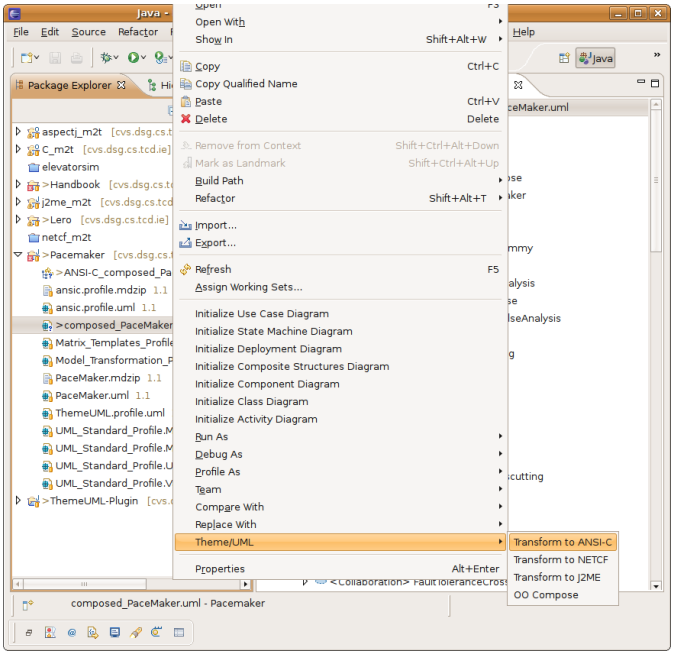

Figure 4: How to chose the option to transform the PIM to a PSM one

The process of transforming the platform independent UML model to a specific one, allows the model to be able to be transformed to source code. This means that the UML datatypes are replaced with the available datatypes of the target platform. On the other hand, there are “things” that can be done in UML, but are not available on every target platform. In this case, those constructs have to be replaced with something that can be done on the target platform. For example, a one to many association can be modeled in UML, but has to be replaced with some kind of a container on most target platforms.

For each supported target platform there exist a UML profile. The datatypes and container constructs defined there will be the replacement for the UML ones in the platform independent model. To adapt to another target platform and / or to modify the existing functionally the plugin has to be modified.

3.3 Code-Generation

The code generation is provided in form of a couple of OpenArchitectureWare eclipse projects. Each of those projects is responsible for one specific target platform.

To generate the programing code for a target platform, it is necessary to import the corresponding OpenArchitectureWare project to eclipse.

The second step is to copy the PSM XML-file to the root folder of the OpenArchitectureWare project in eclipse, see Figure 5.

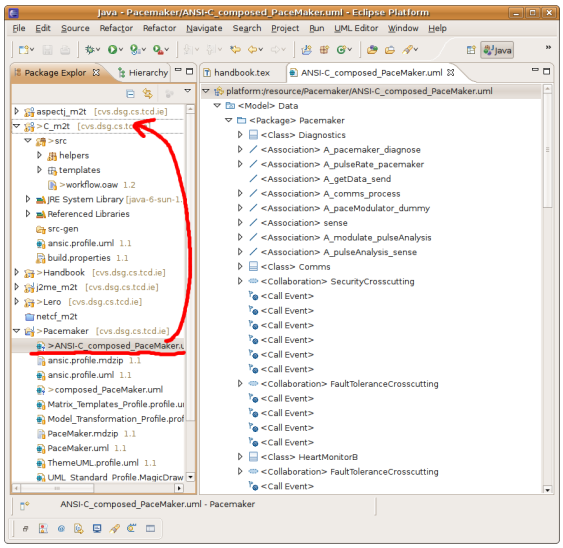

Figure 5: Copy the PSM UML file to the corresponding OpenArchitectureWare project folder

The process of generating the programing code is controlled by the file “workflow.oaw” in the “src” folder. At the tag “modelFile” the “value” has to be set to the file name of the PSM XML-file. See Figure 6 for the right location. This example uses the ANSI-C target platform, for the others the procedure is the same.

If the workflow file is correctly adapted a right click on it and the selection of the option “Runs As/oAW Workflow” starts the workflow. The folder “src-gen” will contain the generated source code.

It is possible to alter the UML file and to rerun the workflow. The new changes will be applied to the source code files. The code for the operations definitions contain special comments, called “PROTECTED REGION” (begin and end). Code inserted between these comments will not be changed by OpenArchitectureWare. This means that is possible to alter the model if necessary and your own code will not be changed by re-generation. See Figure 7 for a code example.

3.4 Limitations

There are some known limitations with the Theme/UML-Plugin. If you encounter a problem that is not mentioned here, please file a bug-report to the mail address `Eamonn.Linehan@cs.tcd.ie`.

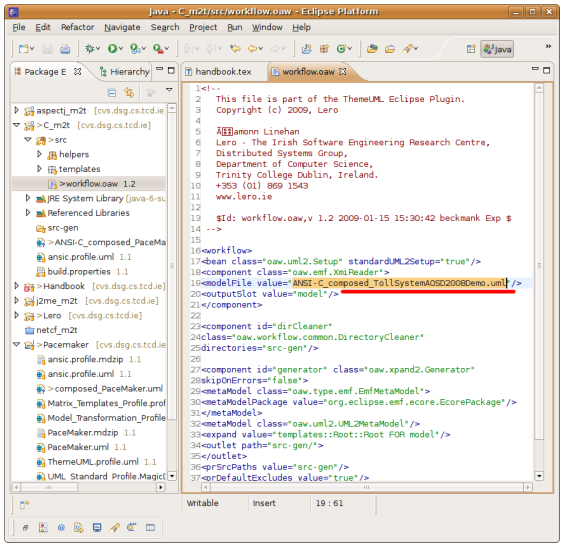

Figure 6: Insert the PSM XML-file name at the OpenArchitectureWare workflow file

```
46
47static  void  _do_sensePulse(HeartMonitor* this)
48{
49
50
51
52
53    /*PROTECTED REGION ID(HeartMonitor_do_sensePulsefalse) ENABLED START*/
54
55     // insert implementation here!
56
57    /*PROTECTED REGION END*/
58
59    return;
60}
```

Figure 7: Protected region to insert your own code

3.4.1 Code-Generation

The Code-Generation has some limitations. One of the main problems is the usage of the sequence diagrams to generate the right method calls inside an operation. We found that the UML model was the source some problems. Specifically, if a sequence diagram is exported by MagicDraw (every version including the current v16) the chronological information of the method calls and returns is lost.

The following caveats are known regarding the code generation from the sequence diagrams:

- Only one call of an operation on one single liveline from the outside (gate?) of the system is possible. See picture 8 for a good and a bad sequence diagram example.
- There should be only two livelines in one sequence diagram. If there are more, the generated code might be wrong and have to be manually adapted.
- The generation process does not know how the parameter of an operation and the variables inside the code are intended to be used. Therefore for each call of an operation all parameters will be declared as local variables (This is more a feature, than a problem).
- The order of other operation calls within one operation can be wrong. You can reduce the chance of this happening by taking care, that the operation calls are in the right order in the MagicDraw application. See Figure 9 for the parts that need attention.

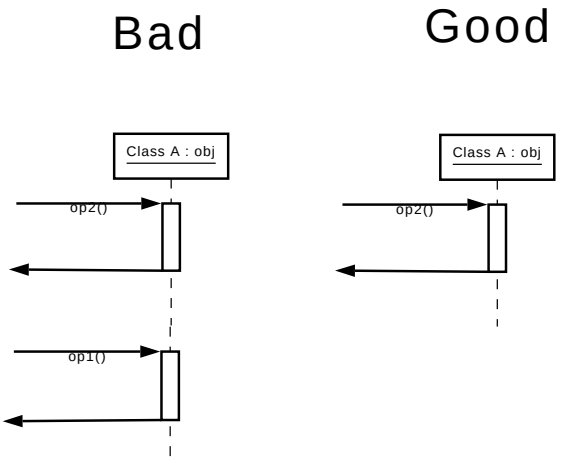

Figure 8: Bad/Good sequence diagrams. Only one call from outside of the system is possible.

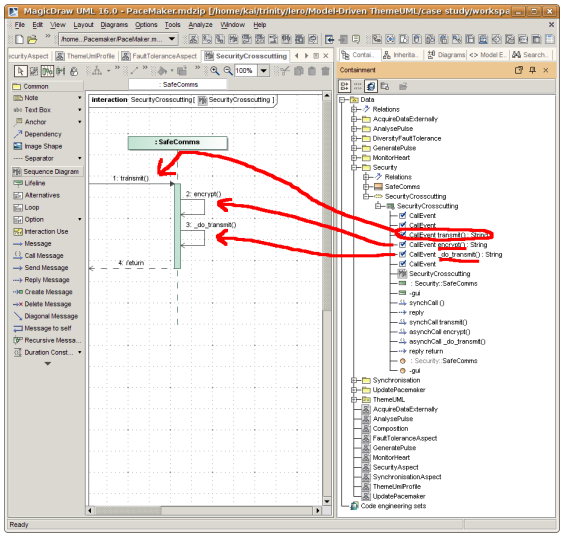

Figure 9: The operation calls in MagicDraw have to be in the right order.

3.4.2 ANSI-C Code-Generation

Not supported:

- UML Template classes
- Enums

4 OOP-Model for C

ANSI-C has no language support for object oriented programing. But it is possible, with some workarounds. The right style guide is important.

One problem with OOP in C is, that many things have to be built manually. This can be a source of errors. With code generation, the most of there errors can be avoided. Making OOP with C more usable.

The Theme/UML-Plugin supports the following features:

- Class structures, with public variables.
  - A class is represented by a C-struct.
  - A instance of the structure is a object of the class.
  - One class is implemented in two files (Header and Implementation).
  - The UML-package structure is mapped to different source folders.
  - Private variables are only private by agreement (no access protection).

- Single inheritance
- Access over parent class reference to objects (polymorphism).
- Public and private operations
  - Public operations are over-writable
- Operations are called by a function pointer in the class structure.
- The first parameter is always a pointer to the instance of the class structure.
- Constructor and Destructor for dynamic memory allocation. Init- and Cleanup-Function for static memory management.
- Associations / Aggregations
  - one2one by single pointer or structure member.
  - one2X Array of pointer or variables (Association and Aggregation)
  - one2many Replaced by container system. Simple double Linked-List. Done in the PIM2PSM, can be adapted.